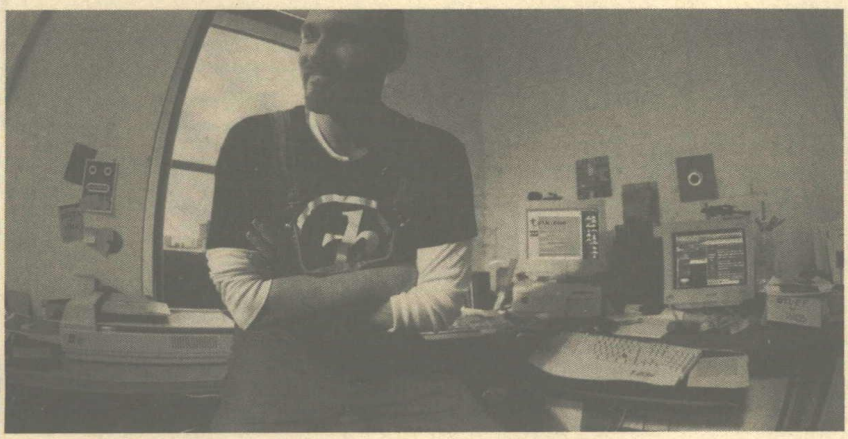
Unix

ไพโรจน์ ไววานิชกิจ

ก้าวแรก... สู่โลก UNIX ตอนที่ 5



ครอบคลุมถึงเรื่องของการเปลี่ยนทิศทาง เรื่องของไปป์ การฟิลเตอร์ การทำงานแบบหลังฉาก และเทคนิคการทำงานกับอักขระต่าง ๆ

บทความในตอนนี้จะกล่าวถึงความสามารถในการทำงานของเชลล์ในระบบปฏิบัติการ UNIX ต่อจากตอนที่แล้ว ซึ่งเนื้อหาจะครอบคลุมถึงเรื่องของการเปลี่ยนทิศทาง เรื่องของไปป์ การฟิลเตอร์ การทำงานแบบหลังฉาก และเทคนิคการทำงานกับอักขระต่าง ๆ

การเปลี่ยนทิศทาง (Redirection)

ตามที่ได้กล่าวถึงในบทความตอนที่แล้วว่าโดยทั่วไปเชลล์ของระบบปฏิบัติการ UNIX จะกำหนดให้อินพุตและเอาต์พุตมาตรฐานเป็นอุปกรณ์เทอร์มินัล โดยเชลล์จะรับอินพุตจากคีย์บอร์ดมาทำการประมวลผลและส่งผลลัพธ์ที่ได้ออกไปยังเอาต์พุตซึ่งก็คือจอภาพแสดงผล

อย่างไรก็ตาม ผู้ใช้งานสามารถที่จะทำการเปลี่ยนแปลงอินพุตและเอาต์พุตให้เป็นอย่างอื่นได้ เช่น เป็นไฟล์หรือเป็นอุปกรณ์อื่น ๆ กระบวนการดังกล่าวมีชื่อเรียกว่าการเปลี่ยนทิศทางอินพุตและเอาต์พุต (Redirection) ด้วยความสามารถดังกล่าวนี้เองจึงทำให้ UNIX เป็นระบบปฏิบัติการที่มีศักยภาพในการสนับสนุนการพัฒนาโปรแกรมประยุกต์ต่าง ๆ ได้เป็นอย่างดี ผู้เขียนจะได้กล่าวถึงตัวอย่างและแนวทางในการใช้ความสามารถในการเปลี่ยนทิศทางอินพุตและเอาต์พุตตามลำดับ

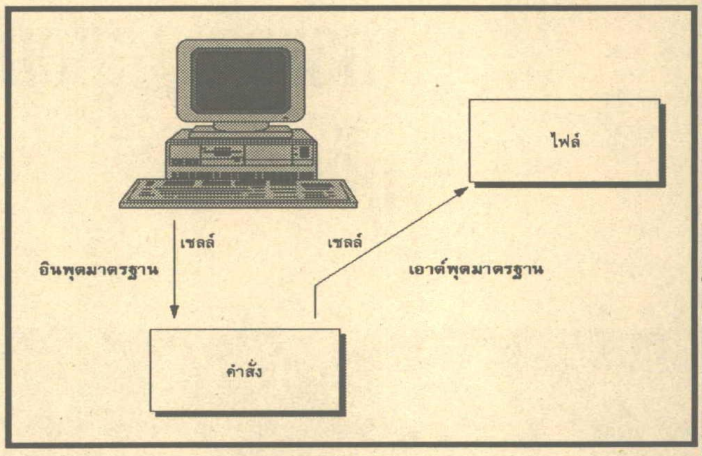
ผู้ใช้งานสามารถทำการเปลี่ยนทิศทางเอาต์พุตได้โดยการใช้เครื่องหมาย ">" ซึ่งจะทำหน้าที่แจ้งกับเชลล์ว่าให้ส่งผลลัพธ์ที่ได้จากการประมวลด้วยคำสั่งหรือโปรแกรมที่ใช้งานอยู่นั้นไปยังไฟล์หรือชุดคำสั่งหรืออุปกรณ์ที่มีชื่ออยู่ต่อจากเครื่องหมาย ">" นั้น รูปแบบของการจัดวางโครงสร้างการเปลี่ยนทิศทางเอาต์พุตเป็นดังนี้

```
command [argument] > filename
```

โดยที่ command ก็คือชุดคำสั่งหรือโปรแกรมที่ต้องการให้เชลล์ทำการประมวลผล โดยอาจจะเป็นโปรแกรมใช้งานทั่วไปหรือยูทิลิตี้ของระบบปฏิบัติการ UNIX สำหรับ argument ก็คืออาร์กิวเมนต์ต่าง ๆ ของคำสั่งที่ใช้งานอยู่นั้น และ filename ก็คือชื่อไฟล์ที่ต้องการให้เชลล์ส่งผลลัพธ์ที่ได้ไปให้ ทั้งนี้ต้องไม่ลืมว่าระบบปฏิบัติการ UNIX มีไฟล์ที่ใช้แทนการอ้างถึงอุปกรณ์ต่าง ๆ อยู่ด้วย (ไฟล์ดีไวซ์) จึงทำให้ผู้ใช้งานมีอิสระในการเลือกจุดหมายปลายทางที่ต้องการให้เชลล์ส่งผลลัพธ์ที่ได้จากการประมวลผลไปให้ รูปที่ 1 แสดงให้เห็นถึงการเปลี่ยนทิศทางเอาต์พุต จะเห็นได้ว่าเอาต์พุตที่ได้รับจากการประมวลผลของเชลล์จะถูกส่งออกไปยังไฟล์ที่ได้รับการระบุจากผู้ใช้งานแทนที่จะส่งออกไปยังอุปกรณ์เอาต์พุตมาตรฐานซึ่งก็คือเทอร์มินัลใช้งาน ขอให้ผู้อ่านลองพิจารณาดูกรอบตัวอย่างต่อไปนี้

```
$ cat > demo.txt
This is the test message typed at the console's keyboard.
The output of this cat command will be redirected to the file named demo.txt.
Next, press CONTROL-D to indicate the end of file.
[CONTROL-D]                              <- กดปุ่ม CTRL - D
$
```

จะเห็นว่าข้อความต่าง ๆ ที่ถูกพิมพ์ผ่านคีย์บอร์ดของอุปกรณ์เทอร์มินัลจะถูกส่งออกไปเขียนยังไฟล์ที่ชื่อว่า demo.txt แทนที่ข้อความเหล่านั้นจะถูกส่งย้อนกลับออกมาแสดงบนหน้าจอดังเช่นในตัวอย่างสุดท้ายของบทความตอนที่แล้ว ซึ่งกระบวนการดังกล่าวนี้เกิดขึ้นจากการใช้เครื่องหมายเปลี่ยนทิศทางให้ส่งผลลัพธ์ที่ได้จากคำสั่ง cat ถูกส่งไปยังไฟล์ที่ชื่อ demo.txt ทั้งนี้ผู้ใช้งานสามารถตรวจสอบได้ด้วยการใช้คำสั่ง **cat demo.txt** เพื่อให้เชลล์แสดงเนื้อหาภายในไฟล์ดังกล่าวสำหรับเปรียบเทียบกับสิ่งที่ได้พิมพ์ไปตามกรอบตัวอย่างข้างต้นนี้



รูปที่ 1 การเปลี่ยนทิศทางเอาต์พุต

```
$ cat stationery
2000 sheets letterhead  ordered: 10/7/94
$ cat tape
1 box masking tape      ordered: 10/14/94
5 boxes filament tape   ordered: 10/28/94
$ cat pens
12 doz. Black pens      ordered: 10/4/90
$ cat stationery tape pens > supply_orders
$ cat supply_orders
2000 sheets letterhead  ordered: 10/7/94
1 box masking tape      ordered: 10/14/94
5 boxes filament tape   ordered: 10/28/94
12 doz. Black pens              ordered: 10/4/90
$
```

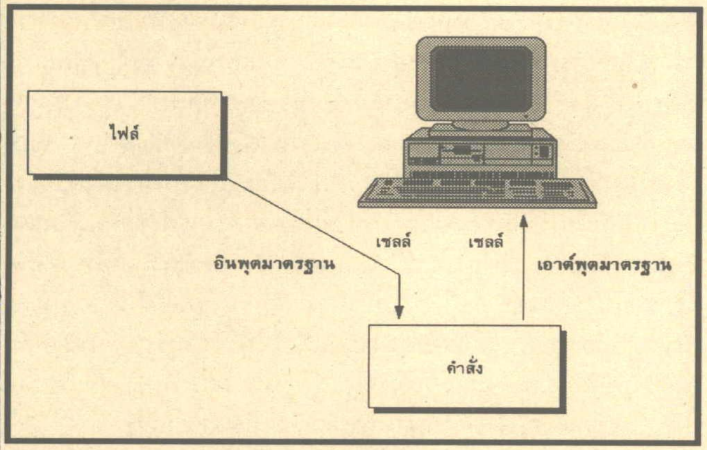
ตัวอย่างในกรอบข้างต้นเป็นแนวทางหนึ่งในการใช้ขีดความสามารถของการเปลี่ยนทิศทางของเชลล์ในระบบปฏิบัติการ UNIX เพื่อประโยชน์ในการจัดการระบบข้อมูลของผู้ใช้งาน ซึ่งจะเห็นว่าสามารถใช้การเปลี่ยนทิศทางนำผลลัพธ์ที่ได้จากคำสั่ง **cat** ที่กระทำกับไฟล์ที่อ้างถึงแต่ละไฟล์มารวมกันออกมายังอุปกรณ์เอาต์พุตมาตรฐานซึ่งก็คือไฟล์ใหม่ที่มีชื่อว่า supply_orders โดยที่เนื้อหาของไฟล์ดังกล่าวจะเกิดขึ้นจากการรวมเนื้อหาของไฟล์ย่อย ๆ ที่ถูกอ้างถึงทั้ง 3 ไฟล์มาเรียงต่อกันเป็นลำดับ

นอกจากความสามารถในการเปลี่ยนทิศทางเอาต์พุตแล้ว เชลล์ของระบบปฏิบัติการ UNIX ยังสนับสนุนการเปลี่ยนทิศทางอินพุต ผู้ใช้งานสามารถกำหนดให้มีการเปลี่ยนแปลงทิศทางของอินพุตได้โดยการใช้เครื่องหมาย "<" ร่วมกับชุดคำสั่งที่ต้องการใช้งาน โดยที่รูปแบบของการจัดชุดคำสั่งเป็นดังต่อไปนี้

```
command [arguments] < filename
```

จะเห็นว่าโครงสร้างของการจัดชุดคำสั่งจะไม่มีความแตกต่างจากกรณีของการเปลี่ยนทิศทางเอาต์พุตมากนัก ยกเว้นเพียงเรื่องการเปลี่ยนทิศทางเท่านั้น โดยคำสั่ง (command) ที่ต้องการใช้งานจะรับคำสั่งจากไฟล์หรืออุปกรณ์ที่ถูกอ้างแทนด้วยชื่อไฟล์ที่อยู่หลังเครื่องหมาย "<" รูปที่ 2 แสดงถึงการเปลี่ยนทิศทางอินพุตสำหรับส่งไปให้เชลล์ทำการประมวลผล ในที่นี้จะสมมติว่าไม่มีการสั่งเปลี่ยนแปลงเอาต์พุตกับคำสั่งที่ยกขึ้นมาเป็นตัวอย่างแต่ประการใด และจากนั้นจะเป็นตัวอย่างการใช้ความสามารถในการเปลี่ยนทิศทางอินพุต โดยผู้เขียนขอยกตัวอย่างการใช้คำสั่ง **cat** ซึ่งมีการสั่งให้นำอินพุตมาจากไฟล์ supply_orders ที่ถูกสร้างขึ้นจากตัวอย่างในเรื่องของการเปลี่ยนทิศทางเอาต์พุต ข้อสังเกตประการหนึ่งสำหรับตัวอย่างที่นำมาแสดงก็คือ ในกรณีของคำสั่ง cat หรืออีกหลาย ๆ คำสั่งในระบบปฏิบัติการ UNIX เช่น คำสั่ง **lpr, sort** หรือ **grep** ผู้ใช้งานสามารถที่จะอ้างชื่อไฟล์ให้เป็นอินพุตของคำสั่งเหล่านี้ได้โดยกำหนดชื่อไฟล์ให้เป็นอาร์กิวเมนต์ของโปรแกรมหรือชุดคำสั่งนั้น ๆ ได้โดยตรง เช่น **cat supply_orders** หรือ **lpr supply_orders** ซึ่งจะให้ผลเช่นเดียวกันกับการใช้ความสามารถในการเปลี่ยนแปลงอินพุตดังเช่น **cat < supply_orders** หรือ **lpr supply_orders** ทุกประการ โดยในกรณีที่กำหนดชื่อไฟล์เป็นอาร์กิวเมนต์ของชุดคำสั่ง ตัวโปรแกรมหรือคำสั่งนั้นจะเป็นผู้ทำหน้าที่ในการอ่านไฟล์โดยตรง แต่สำหรับในกรณีของการเปลี่ยนทิศทางอินพุต เชลล์จะทำหน้าที่ในการป้อนข้อมูลในไฟล์อินพุตให้เป็นอินพุตมาตรฐานของชุดคำสั่งหรือโปรแกรมนั้น ๆ

```
$ cat < supply_orders
2000 sheets letterhead  ordered: 10/7/94
1 box masking tape      ordered: 10/14/94
5 boxes filament tape   ordered: 10/28/94
12 doz. Black pens              ordered: 10/4/90
$
```



รูปที่ 2 การเปลี่ยนทิศทางเอาต์พุต

ในกรณีที่ผู้ใช้งานต้องการที่จะส่งจดหมายอิเล็กทรอนิกส์ไปยังผู้ใช้งานรายอื่น ๆ ก็สามารถที่จะสร้างไฟล์ที่มีเนื้อหาข้อความที่ตนต้องการจะส่งให้เสร็จเรียบร้อย จากนั้นจึงใช้ความสามารถในการเปลี่ยนทิศทางอินพุต ส่งผ่านไฟล์ที่เขียนขึ้นไปยังโปรแกรม mail ด้วยวิธีดังกล่าว จึงทำให้ผู้ใช้งานมีเวลาในการตรวจสอบข้อความ และมีไฟล์จดหมายเป็นหลักฐานอ้างอิงอีกด้วย ตัวอย่างของวิธีการดังกล่าว เช่น ในกรณีที่ผู้เขียนต้องการส่งจดหมายอิเล็กทรอนิกส์ไปยังผู้ใช้งานที่มี login ว่า somsri ผู้เขียนก็จะทำการเขียนข้อความที่ต้องการจะส่งลงในไฟล์ somsri_out ให้เรียบร้อยก่อน จากนั้นจึงใช้คำสั่ง **mail < somsri_out** ซึ่งก็คือการสั่งให้โปรแกรม mail ส่งจดหมายไปยังผู้รับที่มีชื่อว่า somsri โดยเนื้อความที่ให้ส่งไปจะอยู่ในไฟล์ที่มีชื่อว่า somsri_out ทั้งหมด

ผู้เขียนขอแนะนำให้รู้จักกับเครื่องหมาย ">>" ซึ่งใช้ในกรณีของการเปลี่ยนทิศทางเอาต์พุต โดยเครื่องหมายดังกล่าวถูกใช้งานในกรณีที่ต้องการส่งผ่านเอาต์พุตที่ได้จากชุดคำสั่งหรือโปรแกรมให้ออกไปยังไฟล์ที่ถูกสร้างไว้ก่อนแล้ว และต้องการให้ข้อมูลใหม่ที่ได้ไปเพิ่มเติมจากส่วนท้ายของไฟล์นั้น ตัวอย่างที่ยกมาได้แก่การสร้างไฟล์ที่บรรจุข้อมูลเกี่ยวกับวันเวลาและรายชื่อของผู้ที่ทำการ login เข้าใช้ระบบปฏิบัติการ UNIX โดยในขั้นแรกจะใช้คำสั่ง **date > system_info** เพื่อส่งผ่านข้อมูลเกี่ยวกับวันเวลาที่ได้จากคำสั่ง **date** ให้ไปถูกเขียนลงในไฟล์ system_info จากนั้นก็จะเป็นการใช้คำสั่ง **who** ส่งรายละเอียดของผู้ที่ทำการ login เข้ามาในระบบไปเขียนต่อลงในไฟล์ system_info ทั้งนี้ผู้ใช้งานสามารถตรวจสอบความถูกต้องของข้อความในไฟล์ system_info ได้โดยใช้คำสั่ง **cat**

```
$ date > system_info
$ cat system_info
Sat Nov 15 15:00:00 PDT 1997
$ who >> system_info
$ cat system_info
Sat Nov 15 15:00:00 PDT 1997
admin   console      Nov     15 09:12
pairoj  tty09        Nov     15 14:12
suthep  tty01        Nov     15 13:05
```

ไปป์ (pipe)

ความสามารถอีกประการหนึ่งของเชลล์ต่าง ๆ ในระบบปฏิบัติการ UNIX ก็คือการอนุญาตให้ผู้ใช้งานสามารถนำผลลัพธ์ที่ได้จากการทำงานของคำสั่ง ๆ หนึ่งส่งต่อไปเป็นอินพุตของคำสั่งอีกคำสั่งหนึ่งได้ ซึ่งผู้ใช้งานสามารถทำการส่งผ่านข้อมูลในลักษณะนี้ได้หลาย ๆ ทอด เรียกความสามารถดังกล่าวว่า ไปป์ (pipe) ซึ่งเขียนแทนด้วยเครื่องหมาย "|" ในกรณีของการไปป์ข้อมูลระหว่างคำสั่ง 2 ชุด ก็จะเปรียบเสมือนว่าเป็นการกำหนดให้คำสั่งแรกทำการเขียนผลลัพธ์ที่ได้ลงในไฟล์ชั่วคราวไฟล์หนึ่ง จากนั้นจึงนำไฟล์ดังกล่าวนี้มาเป็นอินพุตของคำสั่งที่สอง สำหรับรูปแบบของการใช้งานไปป์กับคำสั่ง 2 ชุดจะเป็นดังนี้

```
command_1 [arguments] | command_2 [arguments]
```

ชุดคำสั่งและโปรแกรมโดยทั่วไปในระบบปฏิบัติการ UNIX มีความสามารถในการทำงานกับกระบวนการไปป์อยู่แล้ว แต่อย่างไรก็ตามผู้อ่านพึงระลึกไว้ว่ายังมีคำสั่งอีกจำนวนหนึ่งซึ่งยอมรับอินพุตได้เฉพาะจากอุปกรณ์อินพุตมาตรฐานที่เทอร์มินัลใช้งานหรือจากไฟล์ข้อมูลเท่านั้น ในการนำคำสั่งเหล่านั้นมาใช้งานร่วมกับการไปป์จึงต้องนำมาพลิกแพลงตามความเหมาะสม ตัวอย่างเช่น ในกรณีของคำสั่ง **tr** ซึ่งมีรูปแบบการใช้งานเป็น **tr string1 string2** จะทำหน้าที่แปลงกลุ่มอักขระที่ถูกระบุไว้ในตัวแปร string1 ไปเป็นกลุ่มอักขระที่ปรากฏในตัวแปร string2 หากต้องการใช้ความสามารถของคำสั่งดังกล่าวเพื่อช่วยในการแปลงตัวอักษรจากไฟล์ ๆ หนึ่ง ก็จำเป็นต้องส่งผลลัพธ์ที่ได้จากการอ่านไฟล์ด้วยคำสั่ง **cat** ไปเป็นอินพุตของคำสั่ง **tr** ซึ่งในกรณีนี้ผู้ใช้งานสามารถใช้ได้ทั้งความสามารถของการไปป์หรือจากการเปลี่ยนทิศทางอินพุตก็ได้ ดังตัวอย่างต่อไปนี้

```
$ cat test_file
This is the demonstration file.
$ cat test_file | tr isd ISD          <- ใช้ความสามารถของการไปป์
```

หรือ

```
$ tr isd ISD < test_file              <- ใช้ความสามารถของการเปลี่ยนทิศทาง
$ cat test_file
ThIS IS the DemonStratIon fIle.
```

ในกรณีที่ผู้ใช้งานทำการ login เข้าสู่ระบบปฏิบัติการ UNIX และต้องการทราบว่าในขณะนี้มีใครกำลังใช้งานระบบอยู่บ้างก็สามารถทำได้โดยการใช้คำสั่ง **who** ดังที่ได้เคยกล่าวมาแล้ว แต่หากจำนวนผู้ใช้งานทั้งหมดมีมาก และต้องการตรวจสอบเจาะจงเฉพาะผู้ใช้งานรายหนึ่งเท่านั้น ก็จะต้องใช้ความสามารถในการไปป์ร่วมกับคำสั่ง **grep** ซึ่งจะแสดงเฉพาะบรรทัดข้อมูลที่มีข้อความซึ่งตามหลัง **grep** ปรากฏอยู่ ดังตัวอย่างต่อไปนี้

```
$ who | grep 'pairoj'
pairoj  tty09          Nov     15 14:12
```

ความสามารถของเชลล์ในการฟิลเตอร์ (filter) หรือกลั่นกรองข้อมูลนับเป็นผลที่ตามมาจากการใช้ความสามารถของการไปป์ คำจำกัดความของฟิลเตอร์ก็คือกระบวนการประมวลผลข้อมูลโดยใช้การคัดเลือกเฉพาะข้อมูลที่ต้องการออกจากกลุ่มข้อมูลอินพุต หรืออาจเป็นการจัดรูปแบบการจัดเรียงข้อมูลให้เป็นไปตามที่ต้องการ ระบบปฏิบัติการ UNIX มีชุดคำสั่งซึ่งสามารถนำมาทำงานร่วมกันโดยใช้ความสามารถในการไปป์ของเชลล์เพื่อสร้างโปรแกรมหรือกลุ่มของชุดคำสั่งซึ่งสามารถทำงานได้ตามความต้องการของผู้ใช้งาน ลองพิจารณาตัวอย่างชุดคำสั่งต่อไปนี้

```
$ who | sort | lpr
```

จะเห็นว่าคำสั่ง **sort** ทำหน้าที่จัดเรียงข้อมูลที่ได้จากคำสั่ง **who** โดยการจัดเรียงจะอ้างอิงตามอักษรตัวแรกสุดของแต่ละบรรทัด ซึ่งก็คือ **login account** ของผู้ใช้งานแต่ละคน สำหรับผลลัพธ์ที่ได้ก็จะถูกส่งต่อไปเป็นอินพุตของคำสั่ง **lpr** ซึ่งทำหน้าที่ส่งข้อมูลเหล่านี้ออกไปยังเครื่องพิมพ์อีกต่อหนึ่ง จะเห็นได้ว่าด้วยความสามารถของการไปป์ทำให้ผู้ใช้งานระบบปฏิบัติการ UNIX สามารถนำชุดคำสั่งต่าง ๆ มาเชื่อมต่อเข้าด้วยกันเพื่อสร้างคำสั่งพิเศษขึ้นใหม่ได้ตามต้องการ

การทำงานแบบหลังฉาก (Background Mode)

โดยทั่วไปการใช้งานชุดคำสั่งของระบบปฏิบัติการ UNIX จะเป็นไปในลักษณะที่ผู้ใช้งานป้อนคำสั่งนั้น ๆ เพื่อให้เชลล์นำไปปฏิบัติงาน ซึ่งผู้ใช้งานต้องรอจนกว่าชุดคำสั่งนั้นได้รับการประมวลผลและทำงานเสร็จสิ้นลงแล้วจึงจะสามารถป้อนชุดคำสั่งใหม่เข้าไปได้ ในบางครั้งผู้ใช้งานอาจมีความจำเป็นที่จะต้องทำการป้อนชุดคำสั่งให้กับระบบปฏิบัติการ UNIX อย่างต่อเนื่อง และไม่ต้องการที่จะรอคอยการประมวลผลจากชุดคำสั่งบางคำสั่งซึ่งกินเวลาในการประมวลผลนาน ในกรณีนี้ระบบปฏิบัติการ UNIX มีการสนับสนุนการทำงานแบบหลังฉาก หรือ background mode ซึ่งทำให้ผู้ใช้งานไม่ต้องรอให้การประมวลผลชุดคำสั่งนั้น ๆ เสร็จสิ้นก่อนที่จะทำการป้อนคำสั่งต่อไป การทำงานแบบหลังฉากจึงเหมาะสำหรับกำหนดให้กับคำสั่งซึ่งกินเวลาในการประมวลผลนานและผู้ใช้งานไม่มีความจำเป็นที่จะต้องทำการตรวจสอบการทำงานบ่อย ๆ สำหรับงาน ๆ หนึ่งซึ่งได้รับการกำหนดให้เป็นการทำงานแบบหลังฉากมักจะนิยมเรียกกันว่า "โปรเซส (process)"

สำหรับการกำหนดชุดคำสั่งที่ต้องการให้ทำงานแบบหลังฉากทำได้โดยการพิมพ์คำสั่งนั้น (อาจมีอาร์กิวเมนต์ตามมา ซึ่งขึ้นอยู่กับการใช้งาน) แล้วตามด้วยเครื่องหมาย "&" เมื่อทำการป้อนคำสั่งเข้าไปแล้วเชลล์จะแสดงเลขหมายโปรเซส (Process Identity เรียกย่อ ๆ ว่า PID) ซึ่งกำหนดให้กับชุดคำสั่งนั้นกลับมาแจ้งให้กับผู้ใช้งานทราบ ตัวอย่างต่อไปนี้เป็นการป้อนคำสั่งซึ่งต้องการให้เชลล์ทำการส่งข้อมูลที่ได้จากคำสั่ง **ls** ไปออกยังเครื่องพิมพ์ซึ่งกำหนดให้เป็นการทำงานแบบหลังฉาก สังเกตว่าตัวเลข 18890 ซึ่งได้รับการตอบกลับจากเชลล์ก็คือ PID นั่นเอง สาเหตุที่ต้องมีการแจ้งเลขหมาย PID ก็เพื่อให้ผู้ใช้รับทราบว่าคำสั่งที่ตนป้อนให้กับเชลล์นั้นจะถูกอ้างถึงด้วยเลขหมาย PID นั้น ๆ ซึ่งนับว่ามีประโยชน์ต่อตัวผู้ใช้เองหากต้องการยกเลิกโปรเซสดังกล่าว

```
$ ls -l | lrp &
18890
```

ในกรณีที่มีการกำหนดคำสั่งให้ทำงานแบบหลังฉากเป็นจำนวนหลายคำสั่ง และผู้ใช้งานต้องการที่จะตรวจสอบโปรเซสที่กำลังทำงานอยู่ก็สามารถที่จะใช้คำสั่ง **ps** ซึ่งย่อมาจาก process status เพื่อแสดงรายละเอียดของโปรเซสต่าง ๆ ที่มีอยู่ในระบบขณะนั้นออกมา ดังตัวอย่างต่อไปนี้

```
$ tail -f outfile &
1099
$ ps
PID     TT      STAT    TIME    COMMAND
1097    03      S       0:02    sh
1099    03      S       0:06    tail
1133    03      R       0:03    ps
```

จากตัวอย่างข้างต้น ผู้เขียนได้สั่งให้เชลล์รับคำสั่ง **tail -f outfile &** ซึ่งเป็นการสั่งให้เชลล์ทำการแสดงข้อความในบรรทัดสุดท้ายของไฟล์ outfile โดยอาร์กิวเมนต์ -f เป็นการแจ้งให้เชลล์ทำการวนลูปแสดงข้อความในบรรทัดสุดท้ายไปเรื่อย ๆ ซึ่งเนื่องจากการป้อนคำสั่งนี้เป็นการกำหนดให้เชลล์ดำเนินการแบบหลังฉาก ดังนั้นจึงมีการกำหนดเลขหมาย PID ของโปรเซสดังกล่าวเป็น 1099 เมื่อใช้คำสั่ง **ps** แสดงรายการของโปรเซสทั้งหมดที่เกิดขึ้นในระบบจะพบว่ามีอยู่ทั้งสิ้น 3 โปรเซส สำหรับการตีความของผลลัพธ์ที่ได้จากคำสั่ง **ps** เป็นดังนี้ PID หมายถึงเลขที่ของโปรเซสซึ่งได้รับการกำหนดโดยระบบปฏิบัติการ UNIX TT หมายถึงหมายเลขของอุปกรณ์เทอร์มินัลที่ทำการควบคุมการทำงานของโปรเซสนั้น ๆ ในที่นี้จะเห็นได้ว่าผู้เขียนทำการป้อนคำสั่งผ่านทางอุปกรณ์เทอร์มินัลหมายเลข TTY03 สำหรับ STAT เป็นการแสดงสถานะของโปรเซสแต่ละตัวซึ่งโดยทั่วไปมักจะพบว่ามีค่าเป็น S (Sleep : หยุดทำงานไปเป็นเวลาไม่ถึง 20 วินาที) หรือ I (Idle : หยุดทำงานไปเป็นเวลานานกว่า 20 วินาที) หรือ R (Runable : กำลังทำงานอยู่) สำหรับ TIME แสดงถึงระยะเวลานับตั้งแต่โปรเซสเริ่มทำงานจนถึงปัจจุบัน

ในกรณีที่ผู้ใช้งานต้องการยกเลิกการทำงานแบบหลังฉากของโปรเซสใดโปรเซสหนึ่งก็สามารถทำได้โดยการใช้คำสั่ง **kill** ตามด้วยหมายเลข PID ของโปรเซสที่ต้องการยกเลิกนั้น สำหรับการยกเลิกโปรเซสที่ทำงานแบบหลังฉากทั้งหมดก็กระทำได้โดยการใช้คำสั่ง **kill 0**

อักขระพิเศษ (Special Character)

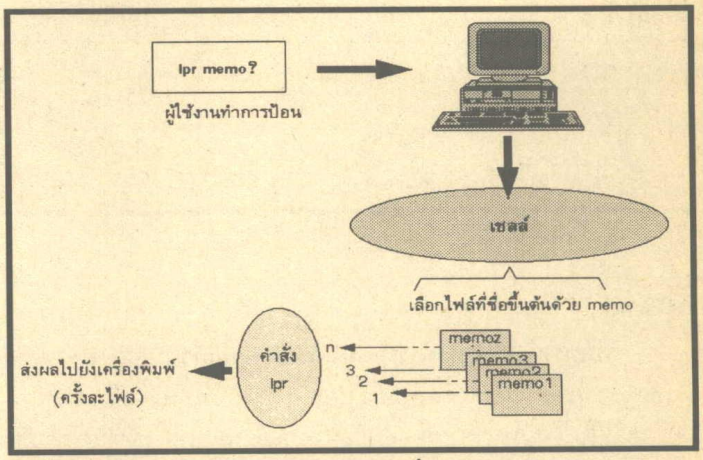
โดยหลักปฏิบัตินิยมทั่วไป การตั้งชื่อไฟล์หรือไดเรคทอรีมักจะกระทำขึ้นโดยใช้กลุ่มอักขระที่อยู่ในชุดอักขระ 26 ตัวในภาษาอังกฤษ ซึ่งก็มีทั้งที่เป็นตัวพิมพ์เล็กและตัวพิมพ์ใหญ่ นอกจากนั้นก็ยังอาจประกอบด้วยเครื่องหมายพิเศษต่าง ๆ เช่น "/" "_" "-" เป็นต้น แต่ทราบกันหรือไม่ครับว่าระบบปฏิบัติการ UNIX ได้รับการออกแบบขึ้นให้สามารถทำงานร่วมกับอักขระพิเศษเพื่อเพิ่มประสิทธิภาพและความสะดวกในการทำงานร่วมกับผู้ใช้งาน ผู้เขียนจะขอปิดท้ายบทความชุดนี้ด้วยเรื่องราวของอักขระพิเศษ 3 ตัวซึ่งผู้อ่านสามารถนำไปเพิ่มประสิทธิภาพในการใช้งานกับระบบปฏิบัติการ UNIX ได้เป็นอย่างดี อักขระพิเศษทั้ง 3 ตัวนั้นก็ได้แก่ "?" "*" และ "[ ]"

อักขระ ?

สำหรับเครื่องหมายคำถามหรืออักขระ "?" นี้จะใช้สำหรับแทนที่อักขระที่ไม่ระบุเจาะจงในส่วนของชื่อไฟล์หรือไดเรคทอรีได้หนึ่งตำแหน่ง เมื่อมีการป้อนคำสั่งโดยอ้างถึงชื่อไฟล์หรือไดเรคทอรีซึ่งมีส่วนใดส่วนหนึ่งของชื่อถูกแทนที่ด้วยเครื่องหมาย "?" เชลล์ของระบบปฏิบัติการ UNIX จะทำการตรวจสอบหาไฟล์หรือไดเรคทอรีที่มีชื่อใกล้เคียงกับชื่อที่ป้อนไปมากที่สุด ทั้งนี้เชลล์จะเปรียบเทียบว่าตำแหน่งของอักขระตัวอื่น ๆ ที่มิได้ถูกแทนที่ด้วยเครื่องหมาย "?" ทั้งของชื่อไฟล์ที่ผู้ใช้ได้ป้อนไว้และของไฟล์หรือไดเรคทอรีที่ปรากฏอยู่ภายใต้ไดเรคทอรีใช้งานในปัจจุบันนั้นตรงกันทั้งหมด จากนั้นเชลล์จะเลือกรายชื่อไฟล์หรือไดเรคทอรีทั้งหมดที่มีอักขระใด ๆ ก็ได้ตรงกับตำแหน่งของเครื่องหมาย "?" ลองพิจารณาตัวอย่างการใช้อักขระพิเศษด้วยคำสั่งต่อไปนี้

```
$ lpr memo?
```

จากคำสั่งดังกล่าวหมายความว่าให้เชลล์ทำการพิมพ์ข้อความในไฟล์ที่มีชื่อขึ้นต้นด้วยอักขระ 4 ตัวแรกว่า memo ส่วนอักขระตัวสุดท้ายจะเป็นอะไรก็ได้ กระบวนการในการประมวลผลของระบบปฏิบัติการ UNIX สำหรับคำสั่งนี้แสดงในรูปที่ 3 โดยเมื่อมีการป้อนชุดคำสั่งดังกล่าวแล้ว เชลล์จะยังไม่สั่งการให้คำสั่ง **lpr** ทำงานในทันที แต่เชลล์จะทำการตรวจสอบหาไฟล์ที่มีชื่อขึ้นต้นด้วยคำว่า memo และมีความยาว 5 ตัวอักษรทั้งหมดที่มีอยู่ในไดเรคทอรีใช้งานปัจจุบัน เมื่อครบทั้งหมดแล้วจึงจะทำการส่งไฟล์แต่ละไฟล์ที่เลือกไว้ผ่านไปยังคำสั่ง **lpr** เพื่อทำการพิมพ์



รูปที่ 3 กระบวนการทำงานของเชลล์กับคำสั่ง lpr memo?

ข้อความภายในไฟล์ออกไปยังเครื่องพิมพ์จนครบทุกไฟล์ ในกรณีที่ไดเรคทอรีใช้งานปัจจุบันไม่มีไฟล์ที่มีชื่อตรงกับที่ต้องการเลย เชลล์จะทำการส่งชื่อไฟล์ตามที่ผู้ใช้ทำการป้อนซึ่งก็คือคำว่า "memo?" ไปให้กับคำสั่ง **lpr** ซึ่งจะส่งผลให้มีการแจ้งความผิดพลาดว่าไม่พบไฟล์ที่ต้องการกลับมายังผู้ใช้งาน

นอกจากจะอ้างถึงชื่อไฟล์โดยใช้อักขระ "?" วางไว้ท้ายชื่อไฟล์แล้ว ผู้ใช้งานยังสามารถนำอักขระดังกล่าวไปวางไว้ในตำแหน่งใด ๆ ภายในชื่อไฟล์ที่ต้องการอ้างถึงได้ด้วย สำหรับตัวอย่างอีก 2 ตัวอย่างซึ่งแสดงในกรอบข้างล่างนี้จะเป็นการสาธิตให้เห็นถึงการใช้ความสามารถของเชลล์ในการแทนค่าอักขระพิเศษ "?" สิ่งที่ผู้เขียนขอฝากไว้ประการหนึ่งก็คือ อักขระ "?" นี้จะใช้แทนตัวอักขระได้เพียงหนึ่งตัวเท่านั้น หากต้องการแทนตัวอักขระมากกว่าหนึ่งตัวก็จะต้องเพิ่มจำนวนอักขระพิเศษนี้ขึ้นให้เท่ากับจำนวนของอักขระภายในชื่อไฟล์ที่ต้องการค้นหา ซึ่งเมื่อผู้เขียนได้กล่าวถึงอักขระพิเศษ "*" ก็ขอให้ผู้อ่านลองเปรียบเทียบความแตกต่างในการใช้งานด้วย ทั้งนี้เพื่อให้เป็นประโยชน์ต่อตัวผู้อ่านทุกท่านในการที่จะเลือกใช้อักขระพิเศษเหล่านี้ให้ตรงกับความประสงค์ในการใช้งานของตน

ตัวอย่างที่ 1

```
$ ls
mem      memo12  memo9   memopairoj      newmemo5
memo     memo5   memoa   memos
$ ls memo?
Memo5    memo9   memoa   memos
```

ตัวอย่างที่ 2

```
$ ls
7may4report     may14report     may4report.79   mayqreport
may.report      may4report      may_report      mayreport
$ ls may?report
may.report      may4report      may_report      mayqreport
```

ตัวอย่างที่ 3

```
$ ls
report01                report02_a      report03
reportaa
report056       report952
$ ls repo?t0?
report01        report03
```

อักขระ *

บทบาทหน้าที่การทำงานของอักขระพิเศษตัวนี้มีความคล้ายคลึงกับอักขระพิเศษ "?" ที่ได้กล่าวถึงไปแล้ว ต่างกันตรงที่ว่าอักขระพิเศษ "*" จะใช้ในการแทนชื่อหรือกลุ่มอักขระที่อยู่ในตำแหน่งอ้างอิงเดียวกันและที่อยู่หลังจากตำแหน่งของอักขระพิเศษนี้ ซึ่งทั้งอักขระที่ตรงกับตำแหน่งอ้างอิงและอักขระที่อยู่ถัดออกไปนั้นจะหมายรวมไปถึงช่องว่างซึ่งไม่มีอักขระใด ๆ อยู่ด้วย ผู้เขียนขอให้ลองพิจารณาดูตัวอย่างการใช้อักขระพิเศษ "*" โดยอ้างถึงคำสั่ง ls เปรียบเทียบกับในกรณีของอักขระพิเศษ "?" ดังแสดงในกรอบตัวอย่างต่อไปนี้

```
$ ls
mem     memo12  memo9   memopairoj      newmemo5
memo    memo5   memoa   memos
$ ls memo*
memo12  memopairoj      memo    memo5   memoa   memos
```

ประโยชน์ประการสำคัญที่ได้จากการใช้อักขระพิเศษ "*" นี้ก็คือในกรณีที่ผู้ใช้งานทำการจัดเก็บไฟล์ข้อมูลของตนโดยแยกออกเป็นหมวดหมู่ด้วยการใช้การตั้งนามสกุลลงท้ายตามประเภทของข้อมูล เช่น ลงท้ายชื่อไฟล์ด้วย .txt หมายถึงไฟล์เอกสารทั่วไป ลงท้ายด้วย .let หมายถึงไฟล์จดหมาย ในกรณีที่ต้องการจัดการกับไฟล์ทั้งหมดในกลุ่มเหล่านี้ ผู้ใช้งานก็สามารถที่จะใช้การอ้างชื่อไฟล์ด้วยอักขระพิเศษ "*" ได้ ซึ่งจะทำให้ลดการพิมพ์หรือป้อนชุดคำสั่งซ้ำ ๆ ได้ เช่น หากต้องการสั่งพิมพ์ไฟล์เอกสารทั้งหมดที่มีอยู่ในไดเรคทอรีใช้งานปัจจุบัน ก็สามารถทำได้โดยใช้คำสั่ง **lpr *.txt** เป็นต้น

อักขระ [ ]

สำหรับอักขระพิเศษตัวสุดท้ายที่จะขอกล่าวถึงก็คืออักขระ "[ ]" ซึ่งอันที่จริงแล้วน่าจะเรียกว่าเป็นคู่อักขระมากกว่า การใช้อักขระพิเศษตัวนี้ก็เพื่อประโยชน์ในการอ้างชื่อไฟล์ที่มีตัวอักขระตามตำแหน่งที่กำหนดตรงกับกลุ่มตัวอักขระที่ผู้ใช้งานต้องการจะอ้างถึง ทั้งนี้จะเห็นได้ว่ารูปแบบการใช้งานจะคล้ายคลึงกับการใช้อักขระพิเศษ "?" มาก ต่างกันตรงที่ในกรณีของการใช้งานอักขระพิเศษ "?" ก็เพื่อจุดประสงค์ในการค้นหาไฟล์ที่มีตัวอักขระใด ๆ ก็ได้ตามตำแหน่งที่ต้องการอ้างถึง แต่ในกรณีของอักขระพิเศษ "[ ]" จะเป็นการจำกัดการเลือกไฟล์ให้แคบลงอีกโดยให้มีเฉพาะตัวอักขระที่ต้องการเท่านั้น ทั้งนี้ในการอ้างใช้งานก็จะนำอักขระทั้งหมดที่ต้องการอ้างอิงมาใส่ลงไประหว่างเครื่องหมายก้ามปูทั้งสองนั้น ผู้เขียนจะขอยกตัวอย่างการใช้งานอักขระพิเศษ "[ ]" ขึ้นมาสัก 2 ตัวอย่างเพื่อให้เป็นที่พอเข้าใจ

ตัวอย่างที่ 1

```
$ ls
report_111197   report_121197   report_131197   report_171197
report_251197   report_261197
$ ls report_1[127]*
report_111197   report121197    report171197
```

ตัวอย่างที่ 2

```
$ ls
part0   part1   part2   part3   part4   part5
$ lpr part[012345]
$ lpr part[0-35]
```

จากตัวอย่างที่ 1 ผู้เขียนต้องการเลือกไฟล์รายงานที่ได้รับการเขียนขึ้นในวันที่ 11, 12 และ 17 พฤศจิกายน ค.ศ. 1997 ดังนั้นจึงจำเป็นต้องใช้อักขระพิเศษ "[ ]" ในการกำหนดเซตของอักขระที่ผู้เขียนต้องการแทนที่ ซึ่งในกรณีของตัวอย่างแรกนี้จะเป็นตำแหน่งตัวอักษรลำดับที่ 9 โดยจะจำกัดให้แสดงเฉพาะตัวเลข 1, 2 และ 7 เท่านั้น สำหรับส่วนท้ายของชื่อไฟล์นั้นจะเหมือนกันจึงสามารถใช้เครื่องหมาย "*" ต่อท้ายได้เลย ผู้อ่านจึงควรจำไว้ว่าเราสามารถนำอักขระพิเศษต่าง ๆ มาใช้ร่วมกันในการเลือกไฟล์ที่ต้องการได้ ทั้งนี้จะเห็นว่ารูปแบบในการใช้งานร่วมกันนั้นขึ้นอยู่กับความต้องการและจุดมุ่งหมายในการใช้งานของท่านเอง

ตัวอย่างที่ 2 เป็นการเลือกไฟล์ที่ต้องการให้ถูกพิมพ์ออกทางเครื่องพิมพ์ ในกรณีนี้ผู้เขียนไม่ต้องการให้มีการพิมพ์เนื้อหาของไฟล์ part4 ออกไป จึงใช้คุณสมบัติของอักขระพิเศษ "[ ]" อีกครั้ง โดยกำหนดให้ตำแหน่งของตัวอักษรตัวที่ 5 เป็นตำแหน่งที่ใช้ในการคัดเลือกไฟล์ จากตัวอย่างนี้ผู้อ่านจะเห็นได้ว่าสามารถทำการกำหนดกลุ่มเซตของตัวอักษร (ในที่นี้คือตัวเลข) ได้ 2 วิธี วิธีแรกก็เป็นดังเช่นที่ได้ยกตัวอย่างมาก่อนหน้านี้แล้ว สำหรับวิธีที่สองก็คือการใช้เครื่องหมาย "-" ถ้าผู้ใช้งานมั่นใจว่าอักขระทั้งหมดที่ต้องการกำหนดเป็นเซตนั้นอยู่ในลำดับที่ต่อเนื่องกัน ทั้งนี้ผู้ใช้งานสามารถที่จะทำการกำหนดกลุ่มอักขระได้ทั้งชนิดที่เป็นอักษรตัวพิมพ์ใหญ่ อักษรตัวพิมพ์เล็ก หรือตัวเลข เช่น 1-3, d-g, A-Z เป็นต้น และในกรณีที่กลุ่มอักขระที่ต้องการมีการก้าวกระโดด ผู้ใช้งานก็ยังสามารถทำการกำหนดได้โดยวิธีการเขียนดังที่แสดงในตัวอย่างที่สอง ยกตัวอย่างเช่น a-cf-h5-9 เป็นต้น

ผู้เขียนขอยกตัวอย่างสุดท้ายในการใช้อักขระพิเศษ "[ ]" สำหรับอ้างชื่อไฟล์ ในตัวอย่างนี้จะเป็นการสั่งแสดงรายชื่อไฟล์ที่มีชื่อว่า letter0 จนถึง letter35 ภายในไดเรคทอรีใช้งานหนึ่ง ขอให้ศึกษาเทคนิคในการอ้างชื่อไฟล์จากตัวอย่างดูครับ

```
$ ls
letter0 letter1 letter2 letter3 letter4 letter5 letter6
letter7
letter8 letter9 letter10        letter11        letter12
letter13        letter14        letter15
letter16        letter17        letter18        letter19
letter20        letter21        letter22        letter23
letter24        letter25        letter26        letter27
letter28        letter29        letter30        letter31
letter32        letter33        letter34        letter35
letter36        letter37        letter38        letter39
letter40        letter41        letter42        letter43
letter44        letter45        letter46        letter47
letter48        letter49        letter50

$ lpr letter[0-9] letter[12][0-9] letter3[0-5]
letter0 letter1 letter2 letter3 letter4 letter5 letter6
letter7
letter8 letter9 letter10        letter11        letter12
letter13        letter14        letter15
letter16        letter17        letter18        letter19
letter20        letter21        letter22        letter23
letter24        letter25        letter26        letter27
letter28        letter29        letter30        letter31
letter32        letter33        letter34        letter35
```

ก่อนจากกัน ผู้เขียนขอตั้งโจทย์ปัญหาให้ผู้อ่านนำกลับไปคิดเป็นการบ้าน ผู้เขียนขอสมมติว่าในไดเรคทอรีใช้งานปัจจุบันของผู้ใช้งานทุกท่านมีไฟล์อยู่จำนวนหนึ่ง ซึ่งหลังจากใช้คำสั่ง ls แสดงรายชื่อของไฟล์แล้วพบว่ามีรายชื่อดังต่อไปนี้

```
$ ls
intro   notesb  ref2   section1      section3      section4b
notesa  ref1    ref3   section2      section4a
```

อยากทราบว่าผู้อ่านจะใช้คำสั่งอะไรและอย่างไรในการแสดงรายชื่อไฟล์ตามเงื่อนไขต่อไปนี้

1. แสดงรายชื่อทุกไฟล์ที่มีชื่อขึ้นต้นด้วยคำว่า section
2. แสดงรายชื่อไฟล์เฉพาะ section1, section2 และ section3
3. แสดงรายชื่อไฟล์เฉพาะ intro
4. แสดงรายชื่อไฟล์เฉพาะ section1, section3, ref1 และ ref3

ไม่ยากใช่ไหมครับ... ในที่สุดผู้เขียนก็ได้กล่าวถึงรายละเอียดพื้นฐานของเชลล์จนครบแล้ว สำหรับบทความตอนต่อไปจะเป็นเรื่องเน็ตเวิร์กกิงหรือการนำระบบปฏิบัติการ UNIX มาเชื่อมต่อกันเป็นเครือข่ายคอมพิวเตอร์ แล้วพบกันฉบับหน้าครับ

ฟรี ThaiSoft So Sethaputra Dictionary V2.0 ชุดเต็ม

โปรแกรม ThaiSoft So Sethaputra Dictionary V2.0 ราคาปกติ 1,485 บาท คุณสามารถเป็นเจ้าของได้ฟรี ๆ แต่ต้องมีสิ่งแลกเปลี่ยนเล็กน้อย

ง่าย ๆ เพียงแต่คุณเขียนเทคนิคเกี่ยวกับการใช้คอมพิวเตอร์ที่สามารถนำมาประยุกต์ใช้ในปัจจุบัน เพื่อเพิ่มประสิทธิภาพในการทำงานให้กับฮาร์ดแวร์ อุปกรณ์ต่อพ่วง ซอฟต์แวร์ อินเทอร์เน็ต หรืออาจจะเป็นเทคนิคหรือเรื่องราวที่น่าสนใจอะไรก็ได้ที่เกี่ยวข้องกับคอมพิวเตอร์ โดยแต่ละเรื่องนั้น เมื่อพิมพ์ด้วยฟอนต์ขนาดปกติแล้ว จะต้องมีความยาวไม่ต่ำกว่า 1/2 หน้ากระดาษ A4

เมื่อนึกออกแล้วก็ให้รีบเขียนส่งมายังที่อยู่ด้านล่าง หรือจะแนบไฟล์มากับอีเมล์ ส่งมาที่ june@se-ed.com เรื่องที่ผ่านการพิจารณาจะจัดลงตีพิมพ์ในนิตยสารไมโครคอมพิวเตอร์ และคุณจะได้รับโปรแกรม ThaiSoft So Sethaputra Dictionary V2.0 มูลค่า 1,485 บาท ฟรี (1 ชุด ต่อ 1 เรื่อง)

สนใจติดต่อมาได้ที่ **กองบรรณาธิการนิตยสารไมโครคอมพิวเตอร์** บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) 800/43-45 ซอยตระกูลสุข ถนนอโศก-ดินแดง ดินแดง กรุงเทพมหานคร 10320 (วงเล็บมุมซองว่า ThaiSoft) โทร. 248-6280, 642-9800 แฟกซ์ 246-5999, 642-9866